A Planning Tool for Minimizing Reshuffles in Container Terminals

Miguel A. Salido, Oscar Sapena, Mario Rodriguez, Federico Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

One of the more important problems in container terminal is related to the Container Stacking Problem. A container stack is a type of temporary store where containers await further transport by truck, train or vessel. The main efficiency problem for an individual stack is to ensure easy access to containers at the expected time of transfer. Since stacks are 'last-in, first-out', and the cranes used to relocate containers within the stack are heavily used, the stacks must be maintained in a state that minimizes on-demand relocations. In this paper, we present a new domain-dependent planning heuristic for finding the best configuration of containers in a bay. Thus, given a set of outgoing containers, our planner minimizes the number of relocations of containers in order to allocate all selected containers in an appropriate order to avoid further reshuffles.

Figure 1. A container yard (courtesy of Hi-Tech Solutions)

1 Introduction

Loading and offloading containers on the stack is performed by cranes. In order to access a container which is not at the top of its pile, those above it must be relocated. This reduces the productivity of the cranes.

Maximizing the efficiency of this process leads to several requirements. First, each incoming container should be allocated a place in the stack which should be free and supported at the time of arrival. Second, each outgoing container should be easily accessible, and preferably close to its unloading position, at the time of its departure. In addition, the stability of the stack puts certain limits on, for example, differences in heights in adjacent areas, the placement of empty and 'half' containers and so on.

Since the allocation of positions to containers is currently done more or less manually, this has convinced us that it should be possible to achieve significant improvements of lead times, storage utilization and throughput using appropriate and improved techniques.

Figure 1 shows a container yard. A yard consists of several blocks, and each block consists of 20-30 yard-bays [6]. Each yard-bay contains several (usually 6) rows. Each row has a maximum allowed tier (usually tier 4 or tier 5 for full containers). Figure 2 shows a transfer crane that is able to move a container within a stacking area or to another location on the terminal. For safety reasons, it is usually prohibited to move the transfer crane while carrying a container [7], therefore these movements only take place in the same yard-bay.

When an outside truck delivers an outbound container to a yard, a transfer crane picks it up and stacks it in a yard-bay. During the ship loading operation, a transfer crane picks up the container and transfers it to a truck that delivers it to a quay crane.

In container terminals, the loading operation for export containers is carefully pre-planned by load planners. For load planning, a containership agent usually transfers a load profile (an outline of a load plan) to a terminal operating company several days before a ship's arrival. The load pro-

Figure 2. A Rubber-tired gantry crane (courtesy of Kalmar Industries).

file specifies only the container group, which is identified by container type (full or empty), port of destination, and size to be stowed in each particular ship cell. Since a ship cell can be filled with any container from its assigned group, the handling effort in the marshalling yard can be made easier by optimally sequencing export containers in the yard for the loading operation. In sequencing the containers, load planners usually pursue two objectives:

1. Minimizing the handling effort of quay cranes and yard equipment.

2. Ensuring the vessel's stability.

The output of this decision-making is called the ”load sequence list”. In order to have an efficient load sequence, storage layout of export containers must have a good configuration. The main focus of this paper is optimally reallocating outgoing containers for the final storage layout from which a load planner can construct an efficient load sequence list. In this way, the objective is therefore to plan the movement of the cranes so as to minimize the number of reshuffles of containers.

Given a layout, the user selects the set of containers that will be moved to the vessel. Our tool is able to organize the layout in order to allocate these containers at the top of the stacks in order to minimize the number of relocations. Thus a solution of our problem is a layout where all outgoing containers can be available without carrying out any reshuffle.

2 The problem modeled as an AI planning problem

A classical AI planning problem can defined by a tuple $\langle A, I, G\rangle$, where A is a set of actions with preconditions and effects, I is the set of propositions in the initial state, and G is a set of propositions that hold true in any goal state. A solution plan to a problem in this form is a sequence of actions chosen from A that when applied transform the initial state I into a state of which G is a subset.

The container stacking problem is a slight modification of the *Blocks World* planning domain [9], which is a well-known domain in the planning community. This domain consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The positioning of the towers on the table is irrelevant. The *Blocks World* planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower (or on a table). The optimal *Blocks World* planning problem is to do so in a minimal number of moves.

This problem is closed to the container stacking problem, but there are some important differences:

- The number of towers is limited in the container stacking problem: a yard-bay contains usually 6 rows, so it is necessary to include an additional constraint to limit the number of towers on the table to 6.

- The height of a tower is also limited. The effect of limiting the number of levels in a tower on the number of required relocations to reach the goal configuration [8].

- The main difference is in the problem goal specification. In the *Blocks World* domain the goal is to get the blocks arranged in a certain layout, specifying the final position of each block. In the container stacking problem the goal state is not defined as accurately, so many different layouts can be a solution for a problem. The goal is that the most immediate containers to load are in the top of the towers, without indicating which containers must be in each tower.

We can model our problem by using the standard encoding language for classical planning tasks called *PDDL* (Planning Domain Definition Language) [4].

2.1 The container stacking domain

The main elements in a domain specification are (1) the types of objects we need to handle, (2) the types of propositions we use to describe the world, (3) the actions we can perform to modify the state of the world and (4) the function to optimize which is the number of relocations movements. For our proposals, we are going to consider two separate domains, D_1 and D_2, with a different complexity degree. In the first domain D_1, all containers have the same properties and the objective is to place a selected subset of containers, which must be loaded into the next vessel, on top of the stacks. In the second domain D_2, four subsets of containers are selected according to their departure time: *contF, contE, contM, contL*. *contF* is the first set of containers to leave, the next ones are *contE*, then *contM* and the last ones *contL*. In Figure 3 they are identified by means of different colors: red, light gray, dark gray and black, respectively. The red ones (*contF* containers) are the first ones to be loaded into the next vessel and, as it can be observed in the final layout in the figure, they are located on top of the stacks to facilitate their loading.

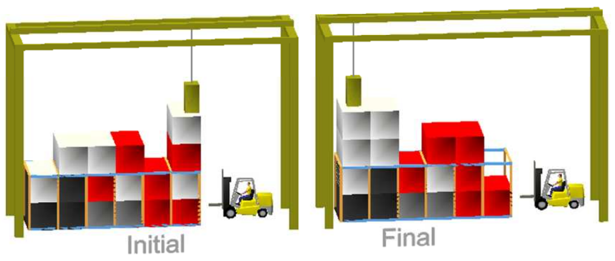

Figure 3. Initial state example (left), and final layout achieved (right) in the domain D_2.

3 A domain-dependent planning heuristic for solving the container stacking problem

Since the container stacking problem can be formalized in *PDDL* format, we can use a general planner to solve our problem instances. Currently we can found several general planners which work well in many different domains, such as *LPG-TD* [3], *MIPS-XXL* [2] and *SGPlan* [1]. However, and due to the high complexity of the domain we are handling, these planners are not able to find good plan solutions efficiently. *LPG-TD*, for example, spends too much time in the preprocessing stages, so it takes a long time to provide a solution. On the contrary, *MIPS-XXL* and *SGPlan* can compute a solution rapidly, but the quality of the obtained solution is not good enough, including some additional relocation movements to achieve the goal configuration.

We have implemented a local search domain-independent planner which can solve quite efficiently many problem instances. This planner has several interesting properties for the container stacking problem:

- It is an anytime planning algorithm [10]. This means that the planner can found a first, probably suboptimal, solution quite rapidly and that this solution is being improved while time is available.

- The planner is complete, so it will always find a solution if exists.

- The planner is optimal. It guarantees finding the optimal plan if there is time enough for computation.

Algorithm 1: Pseudo-code of the domain-dependent heuristic function

```
Data: $s$: state to evaluate
Result: $h$, heuristic value of $s$
$h$ = 0;
if $\exists$ $x$ - container / holding($x$) $\in s$ then
    if goal-container($x$) then
        $h$ = 0.1;
    else
        $h$ = 0.5;
    end
end
for each row $r$ in the yard-bay do
    $\Delta$h = 0;
    for $x$ - container / at($x$, $r$) $\in s \wedge$ goal-container($x$) do
        if $\nexists$ $y$ - container / goal-container($y$) $\wedge$ on($y$, $x$) $\in s$ then
            $\Delta$h = max($\Delta$h, numContainersOn($x$));
        end
    end
    $h$ += $\Delta$h;
end
```

This planner follows an enforced hill-climbing [5] approach with some modifications:

- We apply a best-first search strategy to escape from plateaux. This search is guided by a combination of two heuristic functions and it allows the planner to escape from a local minima very efficiently.

- If a plateau exit node is found within a search limit imposed, the hill-climbing search is resumed from the exit node. Otherwise, a new local search iteration is started from the best open node (the one with the best heuristic value).

This planner solves many problem instances but it can take too much time to find a solution in the hardest problems (usually when the number of containers is high and many reshuffles are required to achieve a goal layout). To improve the planning performance we have designed a new heuristic function specific for this domain. This heuristic computes an estimate of the number of container movements that must be carried out to reach a goal state (see

Algorithm 1). Replacing the traditional heuristic function, based on a relaxed planning graph [5], by this domain-dependent heuristic function we outstandingly improve the planner performance: it solves many more problems and finds better quality plans with considerably less search effort. The plan is returned by the planner as a totally ordered sequence of actions that the transfer crane must carry out to achieve our objective.

3.1 Heuristic improvement

The main goal in the container stacking problem is to minimize the number of reshuffles required to reach a valid final layout. However, several different layouts can be usually achieved making the same number of reshuffles and some of them can be more interesting than the rest according to other important questions:

- It can be interesting to minimize the distance of the goal containers to the right side of the yard-bay, where the transfer crane is located. Achieving this we can spend considerably less time during the truck loading operations.
- It also could be interesting to balance the heights of the stacks to increase the containers stability.

These additional optimization functions have been easily incorporated in our planner by defining the heuristic function as a linear combination of two functions: h(s) = $\alpha * h_1(s) + \beta * h_2(s)$, where:

- $h_1(s)$ is the main heuristic function, which estimates the number of movements required to reach the goal layout (outlined in Algorithm 1). Since this is the main optimization function, α value should be significantly higher than β.
- $h_2(s)$ is the secondary function we want to optimize. This can be, for example, the sum of the distances of the selected containers to the right side of the yard-bay, which can be computed as Algorithm 2 shows.

The benefits of using this combined heuristic function can be observed in Figure 4 and Figure 5. In the first one we want only to minimize the number of reshuffles, i.e. $h(s) = h_1(s)$. In the second one, we also want to minimize the distance of the selected containers to the forklift truck, so we have set $h(s) = 9 * h_1(s) + h_2(s)$. As a result, none of the selected containers (the red ones) are placed in the most left rows, reducing the required time to load the truck.

4 Evaluation

In this section, we have evaluated the minimum number of reshuffles needed to allocate all selected containers

Algorithm 2: Pseudo-code to calculate the distance

```
Data: $s$: state to evaluate
Result: $d$, distance value of $s$
$d = 0$;
for each row $r$ in the yard-bay do
    for $x$ - container / at($x$, $r$) $\in s \wedge$ goal-container($x$) do
        $d = d + (numRows(s) - r)$;
    end
end
```

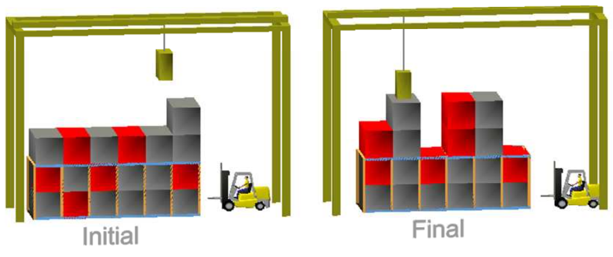

Figure 4. Obtained plan with the initial domain-dependent heuristic.

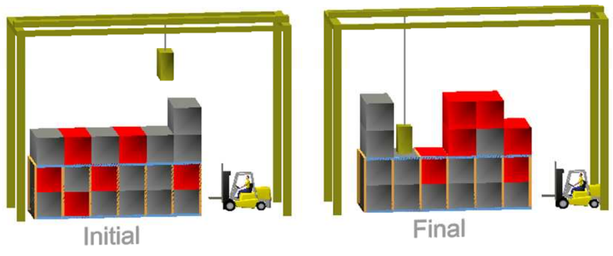

Figure 5. Obtained plan with the distance domain-dependent heuristic.

at the top of the stacks or under another selected containers in such a way that no reshuffles is needed to load outgoing containers.

The experiments were performed on random instances. A random instance is characterized by the tuple $< n, s >$, where n is the number of containers and s is the number of selected containers. Each instance is a random configuration of all containers distributed along the six stack with 4 or 5 tiers. We evaluated 100 test cases for each type of problem.

In Table 1, we present the average running time (in milliseconds) to achieve a solution in both the domain-independent heuristic, based on a relaxed planning graph [5], and our domain-dependent heuristic in problems $< n, 4 >$. Thus, we fixed the number of selected containers to 4 and we increased the number of containers n from 15 to 20. It can be observed that our new domain-dependent heuristic is able to find a solution in a few milliseconds, meanwhile the domain-independent heuristic needs some more time for finding the first solution.

Table 1. Running time in problems $< n, 4 >$.

Instance	A Domain Independent Heuristic	Our New Domain Dependent Heuristic
$< 15, 4 >$	180	6
$< 17, 4 >$	320	10
$< 19, 4 >$	533	15
$< 20, 4 >$	1210	40

In Table 2, we present the average sum of distances between the selected containers and the right side of the layout in both our domain-independent heuristic and our domain-dependent heuristic with distance optimization for problems $< n, 4 >$. As mentioned above, we fixed the number of selected containers to 4 and we increased the number of containers n from 13 to 19. It can be observed that the heuristic for distance optimization helps finding solution plans that place the selected containers closer to the right side of the yard-bay.

Table 2. Avg. distance in problems $< n, 4 >$.

Instance	without Heuristic distance	with Heuristic distance
$< 13, 4 >$	8.15	7.45
$< 15, 4 >$	10.05	8.90
$< 17, 4 >$	10.70	9.55
$< 19, 4 >$	10.85	8.40

5 Conclusions and Further Works

This paper presents the modelling of the container stacking problem form the Artificial Intelligence point of view. We have developed a domain-dependent planning tool for finding an appropriate configuration of containers in a bay. Thus, given a set of outgoing containers, our planner minimizes the number of necessary reshuffles of containers in order to allocate all selected containers at the top of the stacks or under another selected containers in such a way that no further reshuffles are needed to load them.

The initial heuristic function proposed to guide the search has been also improved to allow the optimization of secondary interesting functions as the containers' distance to the right side of the yard-bay, in order to reduce the required time during the loading operations. We have also presented an expanded version of the domain which allow to organize all the containers in the yard-bay according to their departure time. As expected, our domain-dependent tool outperforms the existing domain-independent planners, allowing to obtain high quality plans in few milliseconds.

In further works, we will focus our attention in the development of a more complex domain-dependent planning heuristic to manage new hard and soft constraints.

Acknowledgment

This work is partially supported by the research projects TIN2007-67943-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER), P19/08 (Min. de Fomento, Spain-FEDER), ACOMP/2009/178 Generalitat Valenciana, and by the Technical University of Valencia.

References

[1] Y. Chen, C.W. Hsu, and B.W. Wah, 'SGPlan: Subgoal partitioning and resolution in planning', *IPC-4 Booklet (ICAPS)*, (2004).

[2] S. Edelkamp, 'Taming numbers and durations in the model checking integrated planning system', *Journal of Artificial Intelligence Research (JAIR)*, **20**, 195–238, (2003).

[3] A. Gerevini, A. Saetti, and I. Serina, 'Planning through stochastic local search and temporal action graphs in LPG', *Journal of Artificial Intelligence Research (JAIR)*, **20**, 239–290, (2003).

[4] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, 'PDDL - the planning domain definition language', *AIPS-98 Planning Committee*, (1998).

[5] J. Hoffman and B. Nebel, 'The FF planning system: Fast planning generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[6] Kap Hwan Kim, Young Man Park, and Kwang-Ryul Ryu, 'Deriving decision rules to locate export containers in container yards', *European Journal of Operational Research*, **124**, 89–101, (2000).

[7] Yusin Lee and Nai-Yun Hsu, 'An optimization model for the container pre-marshalling problem', *Computers & Operations Research*, **34**(11), 3295 – 3313, (2007).

[8] M.A. Salido, O. Sapena, M. Rodriguez, and F. Barber, 'A planning-based approach for allocating containers in maritime terminals.', *CAEPIA'09: Thirteenth Spanish Conference for Artificial Intelligence*, (2009).

[9] J. Slaney and S. Thibaux, 'Blocks world revisited', *Artificial Intelligence*, **125**, 119–153, (2001).

[10] S. Zilberstein and S.J. Russell, 'Optimal composition of real-time systems', *Artificial Intelligence*, **82**(1-2), 181–213, (1996).